/\*

CTEs - Common Table Expresions

\*Resultado temporal tomado de una sentencia SQL

\* Es una manera de crear tablas temporales para consultar datos en lugar de utilizar subqueries en una clausula FROM

\* Las CTEs son una alternativa a las subqueries

\* A diferencia de la subqueries, las CTE pueden ser referenciadas multiples veces desde multiples partes de una misma sentencia SQL

\* Mejora la legibilidad de las sentencias

\* El ciclo de vida de las CTEs es el mismo que el de una sentencia SQL

\* Se utilizan en conjunto a las funciones ventana

\* 2 tipos<. No recursiva, recursiva

Sintaxis:

WITH cte\_name (column\_list) AS (

Cte<-query\_definition

)

Statement;

\*/

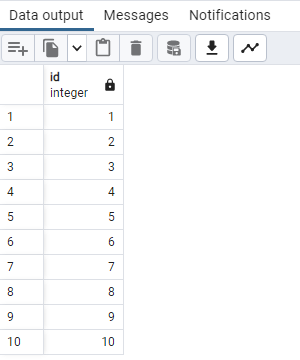
Codigos

WITH numbers AS (

SELECT \* FROM generate\_series(1, 10) AS id

)

SELECT \* FROM numbers;

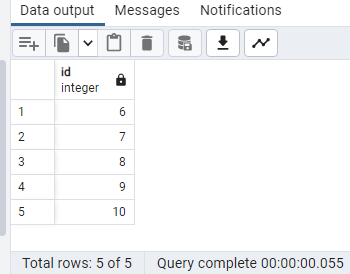


WITH numbers AS (

SELECT \* FROM generate\_series(1, 10) AS id

)

SELECT \* FROM numbers where id > 5;



WITH dates AS (

SELECT \* FROM generate\_series(

'2021-01-01 00:00'::timestamp,

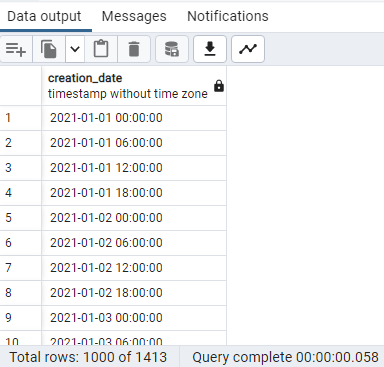
'2021-12-20 00:00',

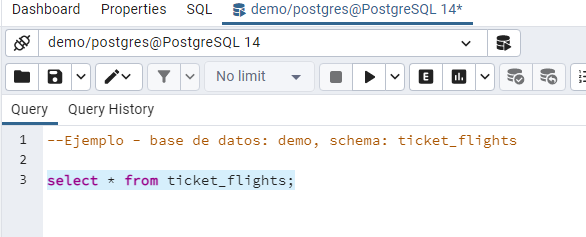
'6 hours'

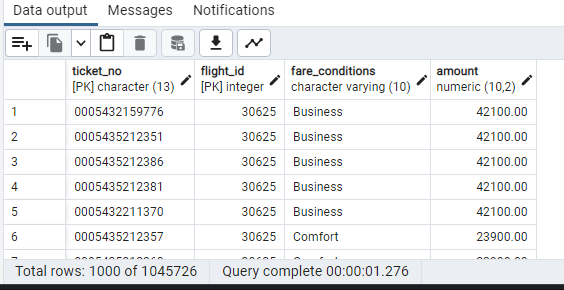
) AS creation\_date

)

SELECT \* FROM dates;







WITH cte\_flights AS (

SELECT flight\_id, fare\_conditions, amount, (

CASE

WHEN amount < 10000 THEN 'CHEAP'

WHEN amount < 30000 THEN 'MEDIUM'

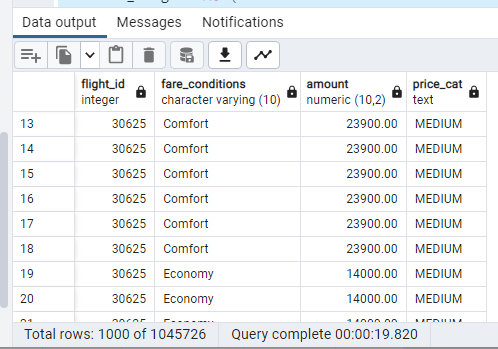
ELSE 'EXPENSIVE'

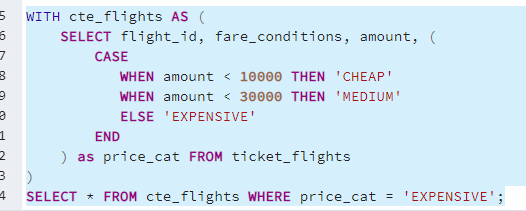
END

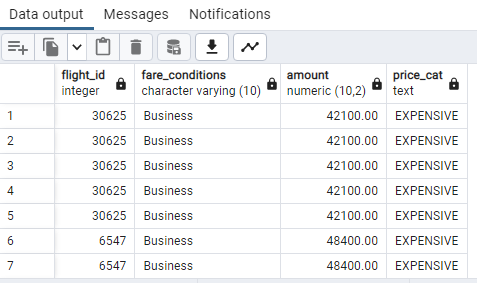
) as price\_cat FROM ticket\_flights

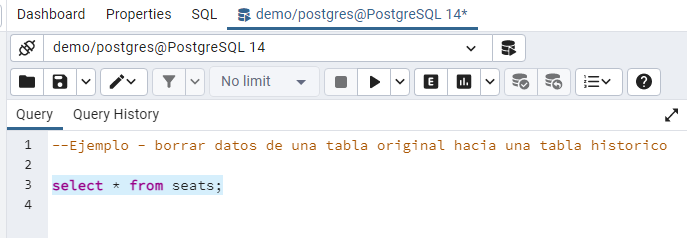
)

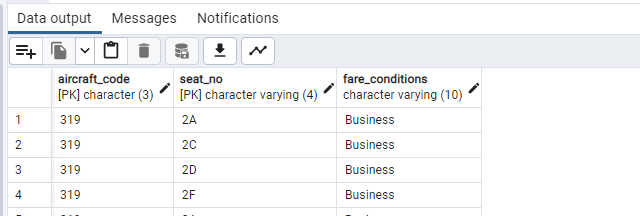
SELECT \* FROM cte\_flights;

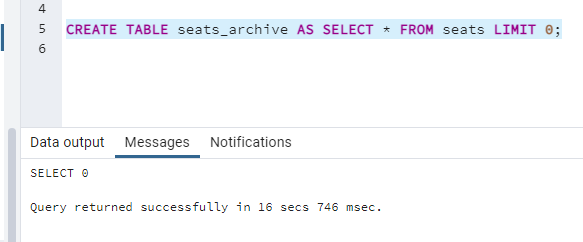


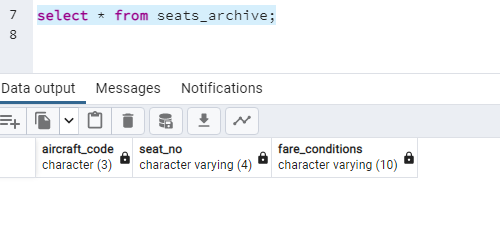




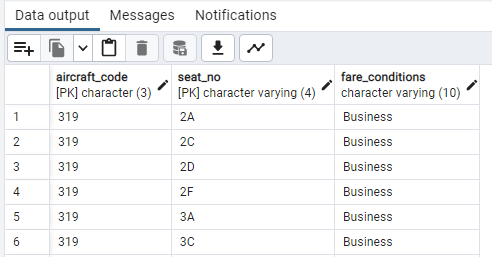












Generamos otra table sin cod 319

WITH cte\_seats\_archive\_aircraft AS (

DELETE FROM seats

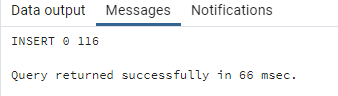
WHERE aircraft\_code = '319'

RETURNING \*

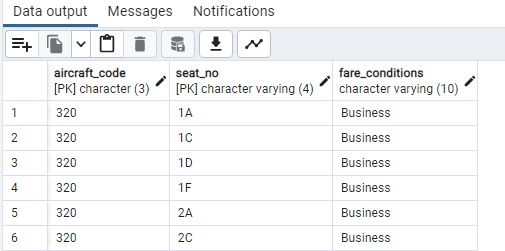
)

INSERT INTO seats\_archive

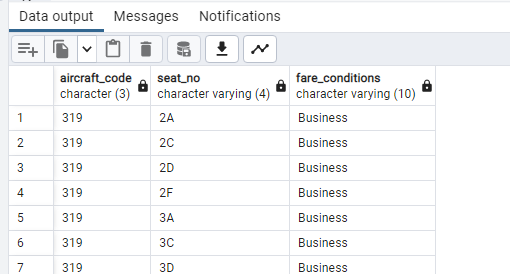
SELECT \* FROM cte\_seats\_archive\_aircraft;











**--Ejemplo - borrar datos de una tabla original hacia una tabla historico**

**select \* from seats;**

**select \* from seats\_archive;**

**drop table if exists seats;**

**drop table if exists seats\_archive;**

**CREATE TABLE seats\_archive AS SELECT \* FROM seats LIMIT 0;**

**INSERT INTO seats\_archive**

**SELECT \* FROM seats;**

**WITH cte\_seats\_archive\_aircraft AS (**

**DELETE FROM seats**

**WHERE aircraft\_code = '319'**

**RETURNING \***

**)**

**INSERT INTO seats\_archive**

**SELECT \* FROM cte\_seats\_archive\_aircraft;**

Funciones agregadas

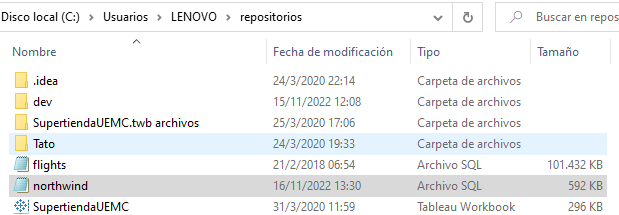
* Las funciones agregadas (COUNT, AVG, SUM …) agregan datos de un conjunto de filas en una sola fila (1 resultado) realizando un cálculo

Funciones ventana

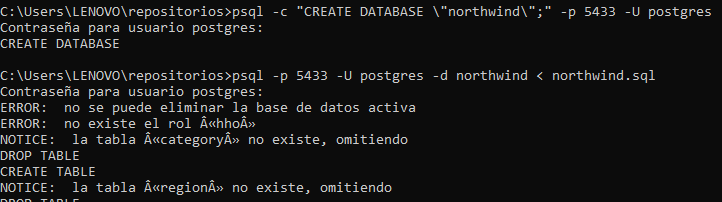
* Las funciones ventana permiten realizar cálculos sobre un conjunto de filas relacionadas con la fila actual.
* No agrupan los datos en un único resultado.
* Permiten realizar cálculos sin perder detalle ni reducir el numero de resultados como ocurre con las funciones agregadas.
* Se crean agregaciones sobre las propias filas, sin reducir el numero de resultados

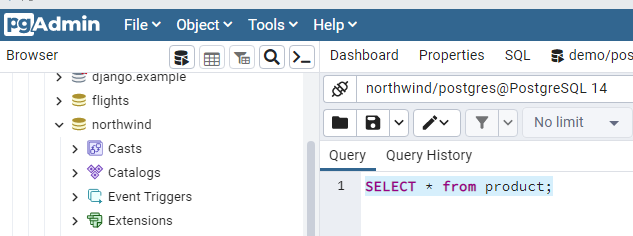
Sintaxis: OVER() PARTITION BY() ODER BY() ROW\_NUMBER() RANK() DENSE\_RANK() FIRST\_VALUE() LAST\_VALUE() LAG() LEAD()

Descargamos base de datos northwind de Github con git clone …. en CMD

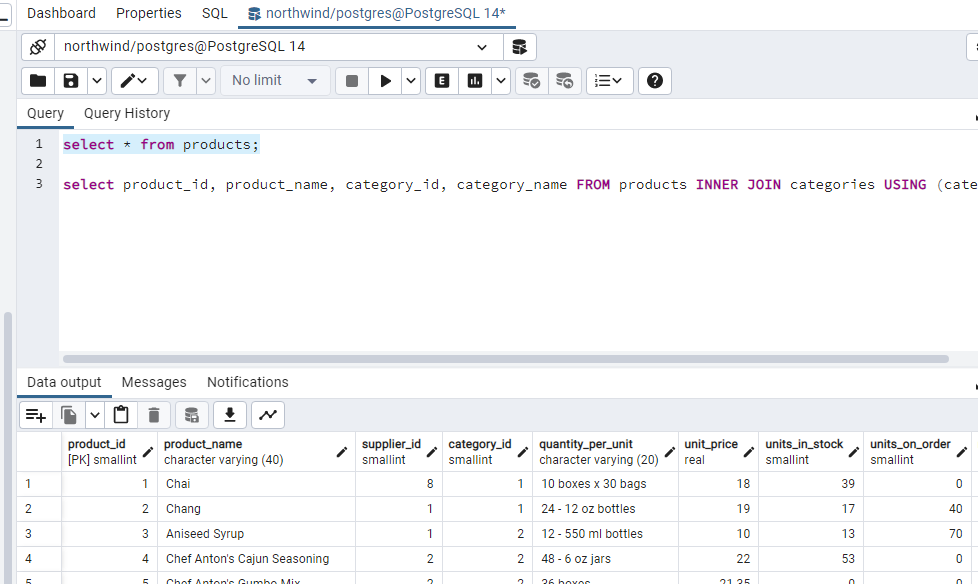


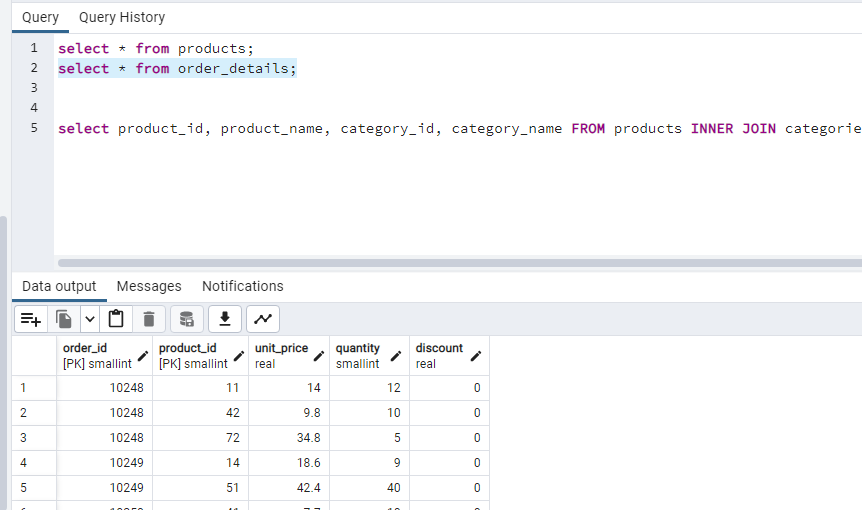
Generamos la BD y la descargamos en Postgres Admin



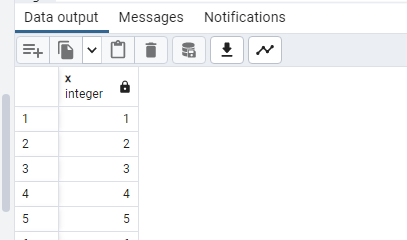


Nos conectamos y generamos consulta

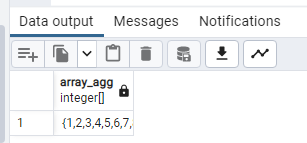




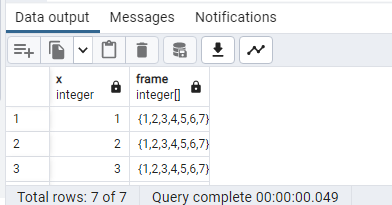




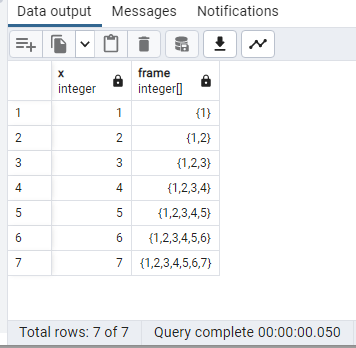


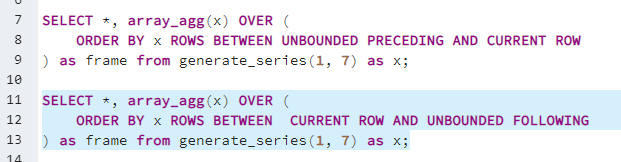


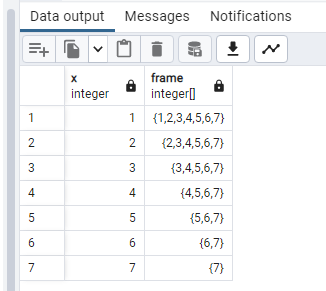




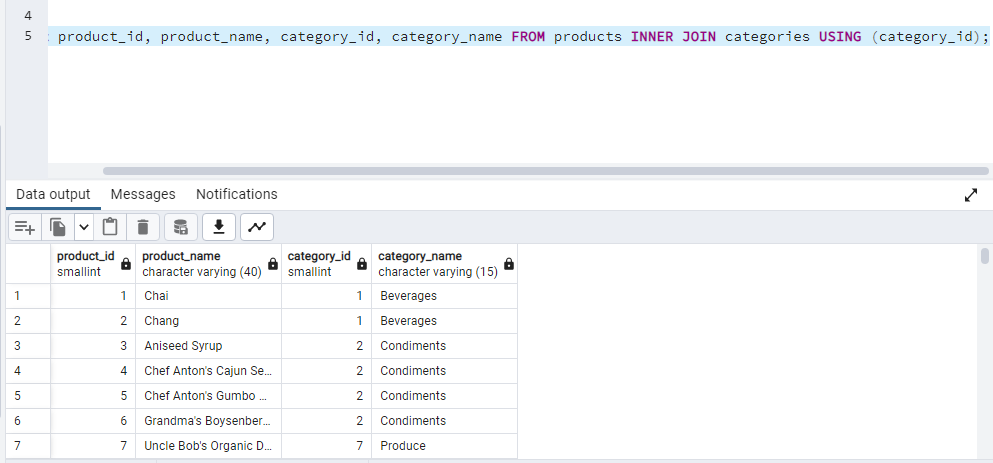








**Consultas base de datos northwind**

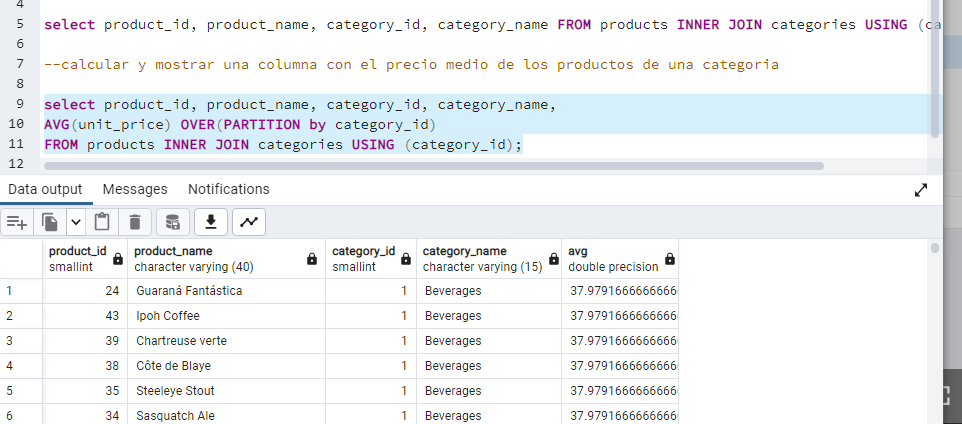


**--calcular y mostrar una columna con el precio medio de los productos de una categoria**

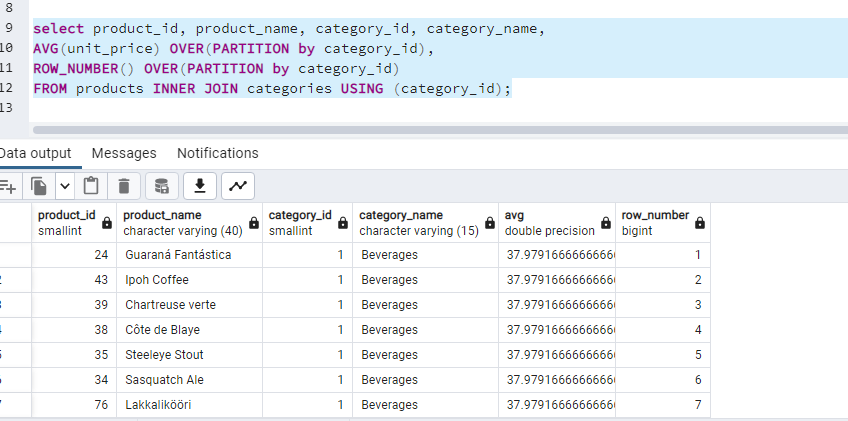
select product\_id, product\_name, category\_id, category\_name,

AVG(unit\_price) OVER(PARTITION by category\_id)

FROM products INNER JOIN categories USING (category\_id);



Ver la cantidad de filas de producto por categoria

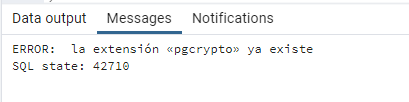


**/\* Cifrado de datos**

**pgcrypto**

**\*/**

CREATE EXTENSION pgcrypto;



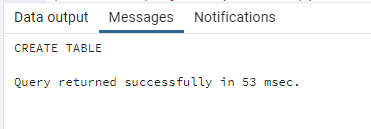
CREATE TABLE ob\_users (

id serial primary key,

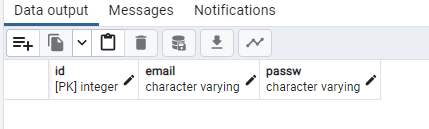
email varchar not null UNIQUE,

passw varchar not null

)



select \* from ob\_users;



**//Inserta datos sin cifrar**

INSERT INTO ob\_users(email, passw) VALUES

('user3@company.com', 'admin'),

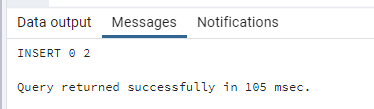
('user4@company.com', '1234')

**//Inserta datos cifrados**

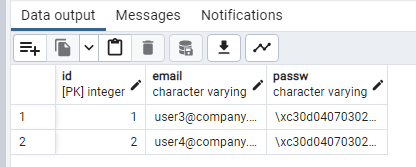
INSERT INTO ob\_users(email, passw) VALUES

('user3@company.com', pgp\_sym\_encrypt('admin', 'secret')),

('user4@company.com', pgp\_sym\_encrypt('1234', 'secret'))

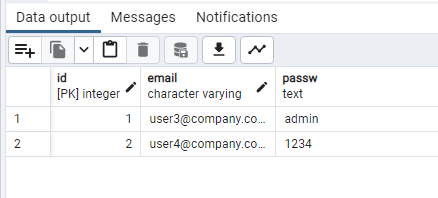






**//Desencripta los datos cifrados**

select id, email, pgp\_sym\_decrypt(passw::bytea, 'secret') as passw FROM ob\_users;



**/\*Particionamiento mediante herencia\*/**

--CREACION de TABLA

create table measurement (

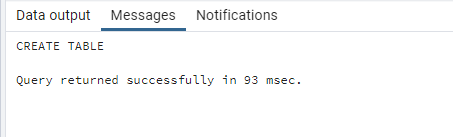
city\_id int not null,

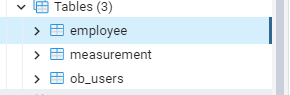
logdate date not null,

peaktemp int,

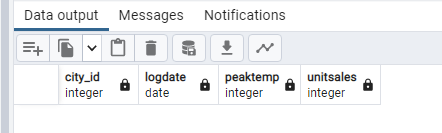
unitsales int

);







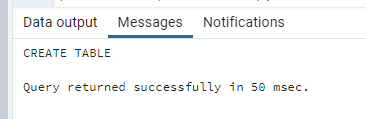


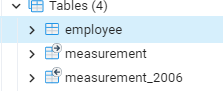
Particionamos y heredamos dos tablas

create table measurement\_2006 (

CHECK (logdate >= DATE '2006-01-01' AND logdate < DATE '2007-01-01')

) INHERITS (measurement);

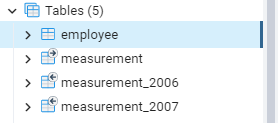




create table measurement\_2007 (

CHECK (logdate >= DATE '2007-01-01' AND logdate < DATE '2008-01-01')

) INHERITS (measurement);



**-- CREAR FUNCION PARA TRIGGER**

**-- Almacenar todos los nuevos registros en la tabla measurement\_2007**

CREATE OR REPLACE FUNCTION measurement\_insert\_trigger()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO measurement\_2007 VALUES (NEW.\*);

RETURN NULL;

END;

$$

LANGUAGE plpgsql;

**-- Ejemplo 2: Detectar el año e insertar en una partición en base al año**

CREATE OR REPLACE FUNCTION measurement\_insert\_trigger()

RETURNS TRIGGER AS $$

BEGIN

IF ( NEW.logdate >= DATE '2020-01-01' AND

NEW.logdate < DATE '2021-01-01' ) THEN

INSERT INTO measurement\_2020 VALUES (NEW.\*);

ELSIF ( NEW.logdate >= DATE '2021-01-01' AND

NEW.logdate < DATE '2022-01-01' ) THEN

INSERT INTO measurement\_2021 VALUES (NEW.\*);

ELSE

RAISE EXCEPTION 'Date out of range. Fix the measurement\_insert\_trigger() function!';

END IF;

RETURN NULL;

END;

$$

LANGUAGE plpgsql;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CREATE TRIGGER insert\_measurement\_trigger

BEFORE INSERT ON measurement

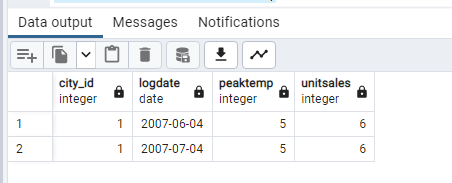
FOR EACH ROW EXECUTE PROCEDURE measurement\_insert\_trigger();

INSERT INTO measurement values

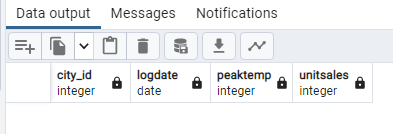
(1, '2007-06-04', 5, 6),

(1, '2007-07-04', 5, 6);

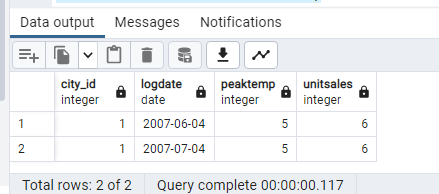
select \* from measurement;



select \* from measurement\_2006;



select \* from measurement\_2007;



EXPLAIN ANALYZE select \* from measurement where logdate = '2007-06-04';

